

ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH



COMPUTER NETWORK

Assignment 1

Implement a streaming video server and client

Teacher: Bùi Xuân Giang

Student¹: Hoàng Gia Khang - 1812535
Trần Danh Hoàng - 1812291
Phan Quốc Long - 1810299
Trần Minh Hoàng - 1812295

Hồ Chí Minh City, 10/2020

¹The contribution of all members is equivalent.



Contents

1	Requirements Analysis	3
1.1	Yêu cầu chức năng :	3
1.2	Yêu cầu phi chức năng :	3
2	Function description	4
2.1	Client :	4
2.2	Server :	4
3	Class diagram	5
4	A Summative Evaluation of Results Achieved	6
5	User manual	7
5.1	Chạy chương trình	7
5.2	Cách sử dụng sau khi chạy chương trình	7
6	Extend	10
6.1	Calculate the statistics about the session	10
6.2	Implement PLAY, PAUSE and STOP buttons	17
6.3	Implement DESCRIBE method	20
7	Reference	22

List of Figures

1	Class Diagram của hệ thống RTP-RTSP Video Streaming	5
2	DESCRIBE in RTSP	6
3	Chạy đoạn code ở terminal thứ nhất	7
4	Chạy đoạn code ở terminal thứ hai	7
5	Cửa sổ giao diện người dùng	8
6	Giao diện khi người dùng bấm Setup	8
7	Giao diện khi người dùng bấm Play	8
8	Giao diện khi người dùng bấm Pause	9
9	Giao diện khi người dùng bấm Teardown	9
10	Giao diện khi bấm vào dấu X	9
11	RTP Stream Analysis	10
12	thời gian đợi mỗi lần gửi tin	11
13	cộng current sequence number thêm 1 khi ListenRtp() được gọi	11
14	Công thức tính Packet loss rate	12
15	Thư viện time và các biến khởi tạo	13
16	Timestart trong playMovie()	14
17	Timeexe and Timeend trong pauseMovie()	14
18	Timeexe and Timeend in pauseMovie()	14
19	countPayload in listenRtp()	15
20	Video data rate formula	15
21	Output with RTP Packet loss rate and Video data rate	16
22	Chương trình tự động SETUP	17
23	Chạy video bằng bấm vào nút PLAY	18
24	Video đang tạm dừng	18
25	Khi bấm STOP sẽ quay trở lại trạng thái ban đầu	19
26	Thoát chương trình	19
27	Viết hàm xử lý yêu cầu Describe từ Client	20
28	Tạo hàm replyDescribe để tạo thông điệp trả về Client	20
29	Thông điệp trả về khi người dùng ấn Describe	21

1 Requirements Analysis

1.1 Yêu cầu chức năng :

- Triển khai một server video trực tuyến và client giao tiếp bằng giao thức RTSP và gửi dữ liệu bằng giao thức RTP
- Client khởi động sẽ mở RTSP socket đến server để gửi yêu cầu cho server
- Trạng thái của Client liên tục cập nhật khi nhận phản hồi từ server

1.2 Yêu cầu phi chức năng :

- Thời gian chờ của datagram socket để nhận RTP data từ server là 0.5s
- Server gửi gói RTP cho client bằng UDP mỗi 50ms

2 Function description

2.1 Client :

- **SETUP:** Gửi yêu cầu setup đến server, đọc phản hồi từ server và phân tích session header để lấy RTSP session ID, tạo một datagram socket để nhận RTP data và đặt thời gian chờ cho socket là 0.5s, video được khởi tạo. yêu cầu SETUP bắt buộc phải được hoàn thành trước khi một yêu cầu PLAY được gửi từ máy client.
- **PLAY:** Gửi yêu cầu play đến server, chỉ ra một dải (range) chỉ rõ một cách cụ thể số hiệu frame bắt đầu được gửi và số hiệu frame kết thúc; đọc phản hồi từ server, video phát.
- **PAUSE:** Gửi yêu cầu pause đến server, đọc phản hồi từ server, video tạm dừng, sau này có thể được nối lại với một yêu cầu PLAY.
- **TEARDOWN:** Gửi yêu cầu teardown đến server, đọc phản hồi từ server, quay trở về trạng thái ban đầu, chưa được SETUP.

2.2 Server :

- Khi server nhận yêu cầu SETUP từ client, server sẽ đáp trả lại bằng các xác nhận các tham số đã được lựa chọn, và điền vào các phần còn thiếu.
- Khi server nhận yêu cầu PLAY từ client, server đọc 1 video frame từ file và tạo RtpPacket. Sau đó, server gửi frame đến client.
- Khi server nhận yêu cầu PAUSE từ client, server sẽ tạm dừng gửi các frame dữ liệu tới máy client.
- Khi server nhận yêu cầu TEARDOWN từ client, server dừng gửi các frame tới máy client và kết thúc một phiên giao dịch của giao thức RTSP.

3 Class diagram

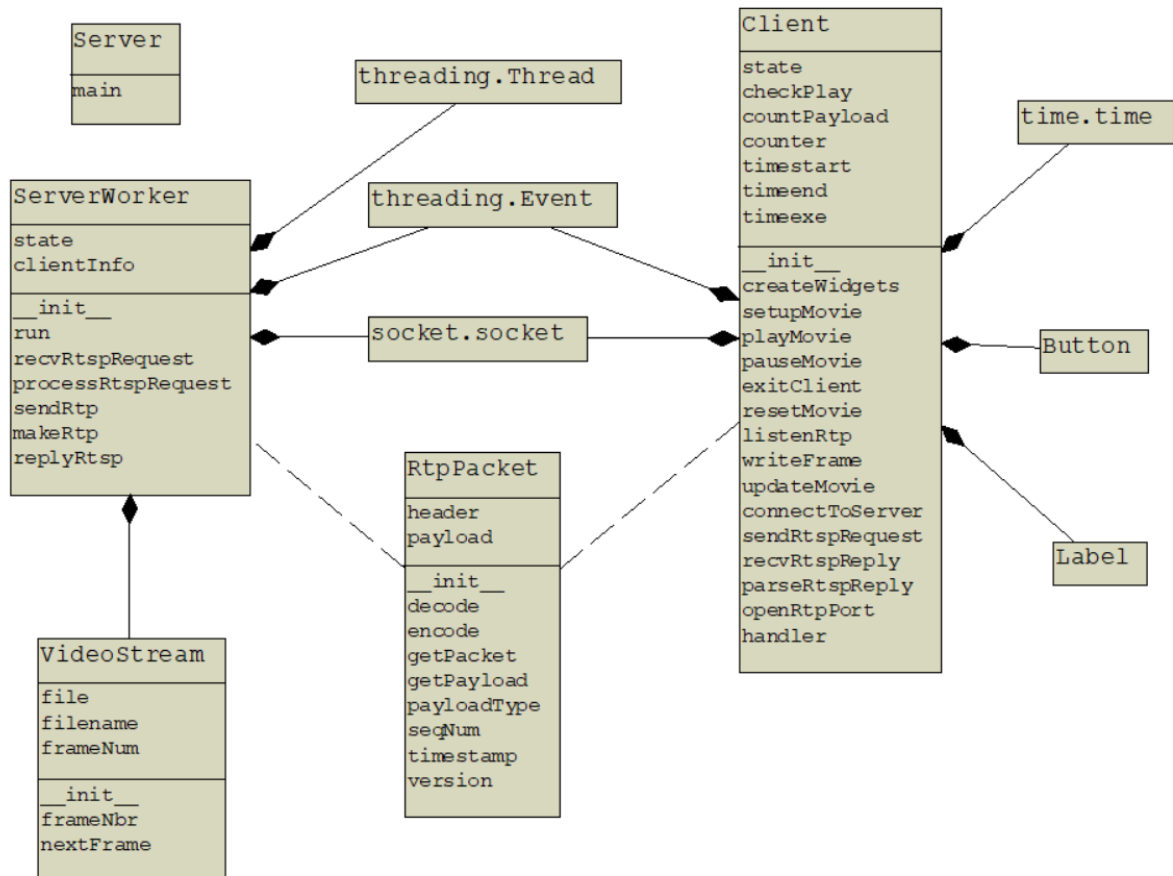


Figure 1: Class Diagram của hệ thống RTP-RTSP Video Streaming

4 A Summative Evaluation of Results Achieved

Chúng ta đã thiết kế được 1 mô hình client-server, với server online ở 1 port cố định (lớn hơn 1024) và được client kết nối để xem video trên 1 port bất kì của client, và Client có 1 GUI có thể SETUP, PLAY, STOP, TEARDOWN video. Chúng ta đã có thể tự biết cách nghiên cứu SPEC của các protocol thông qua các tài liệu RFC, như trong bài tập lớn này là RFC 2326 (RTSP) và RFC 1889 (RTP) để biết cú pháp gửi những thông tin gì cho RTSP Socket, vì phải truyền đúng cú pháp thì bên máy khách mới biết dùng protocol gì để phân giải video, chẳng hạn như bài này phần decode chúng ta mặc định header của UDP là 12 bytes, nên chúng ta chỉ cần lấy payload từ byte thứ 12 trở đi và xử lý video là có thể xem được video rồi, nhưng nếu thực tế với hàng triệu protocol, nếu chúng ta không có thông tin thì sẽ không biết cần lấy từ byte thứ bao nhiêu.

```
104         def pauseMovie(self):
105             """Pause button handler."""
106             if self.state == self.PLAYING:
107                 self.timeend = time.time()
108                 self.timeexe += self.timeend - self.timestart
109                 self.sendRtspRequest(self.PAUSE)
110
```

Figure 2: DESCRIBE in RTSP

Chẳng hạn như bài extend 3, chúng ta có phần describe sẽ dùng "Accept: application/sdp", chúng ta biết sẽ dùng sdp (Session Description Protocol) để phân tích video, và từ SDP chúng ta sẽ biết "kind of stream" là audio(âm thanh) hay video(hình ảnh) hay cả hai. Hay cách decode (h263 với audio và video, L8,L16 với audio). Đó là về phần nghiên cứu, ứng dụng, và hiện thực RTSP Socket

Biết về các trường của RTP packet, sự quan trọng của sequence number khi làm extend 1 để tính packet loss rate và length của payload để tính video data rate.

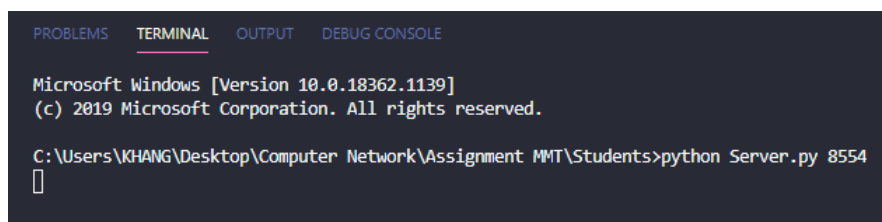
Chúng ta cũng biết cách dịch bit để lấy thông tin từ data của header, ngoài ra còn biết các trạng thái của 1 video (INIT,READY và PLAYING) tùy với trạng thái mà hiện thực những yêu cầu khác nhau.

5 User manual

5.1 Chạy chương trình

Tại folder chứa source code. Mở 2 terminal

- Terminal thứ nhất: `python Server.py server_port`
 - `server_port`: là port mà sever của bạn nghe các kết nối RTSP đến.
 - Port RTSP tiêu chuẩn là 554
 - Nhưng ta phải chọn port > 1024
 - Ví dụ: `python Server.py 8554`



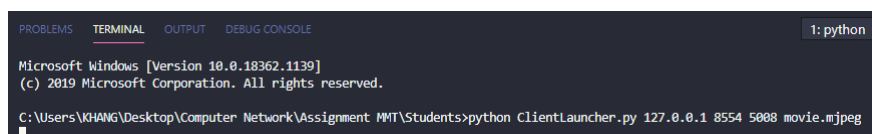
```
PROBLEMS  TERMINAL  OUTPUT  DEBUG CONSOLE

Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\KHANG\Desktop\Computer Network\Assignment MMT\Students>python Server.py 8554
█
```

Figure 3: Chạy đoạn code ở terminal thứ nhất

- Terminal thứ hai: `python ClientLauncher.py server_host server_port RTP_port video_file`
 - `server_host`: tên của máy nơi sever đang chạy (localhost là 127.0.0.1)
 - `server_port`: port server đang nghe (ví dụ trên `python Server.py 8554` thì ở đây `server_port` sẽ là 8554)
 - `RTP_port`: port nơi nhận các gói RTP (ở đây 5008)
 - `video_file`: tên của tệp video bạn muốn yêu cầu (ở đây `movie.mjpeg`)
 - Ví dụ: `python ClientLauncher.py 127.0.0.1 8554 5008 movie.mjpeg`



```
PROBLEMS  TERMINAL  OUTPUT  DEBUG CONSOLE  1: python

Microsoft Windows [Version 10.0.18362.1139]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\KHANG\Desktop\Computer Network\Assignment MMT\Students>python ClientLauncher.py 127.0.0.1 8554 5008 movie.mjpeg
█
```

Figure 4: Chạy đoạn code ở terminal thứ hai

5.2 Cách sử dụng sau khi chạy chương trình

- Sau khi lần lượt thực thi 2 lệnh trên. Cửa sổ giao diện sẽ hiện lên màn hình như ảnh bên dưới



Figure 5: Cửa sổ giao diện người dùng

- Sau đó click vào button Setup để gửi yêu cầu SETUP tới máy chủ

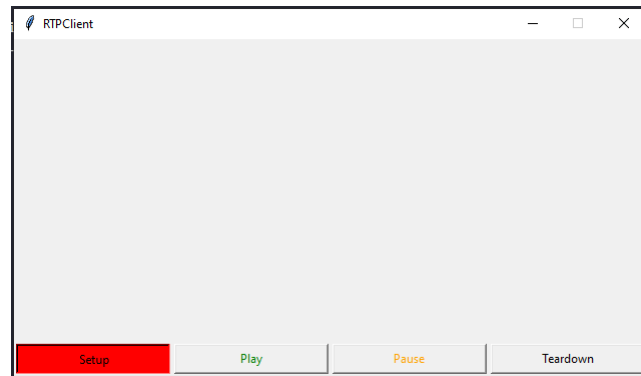


Figure 6: Giao diện khi người dùng bấm Setup

- Bấm vào Play để chạy chương trình

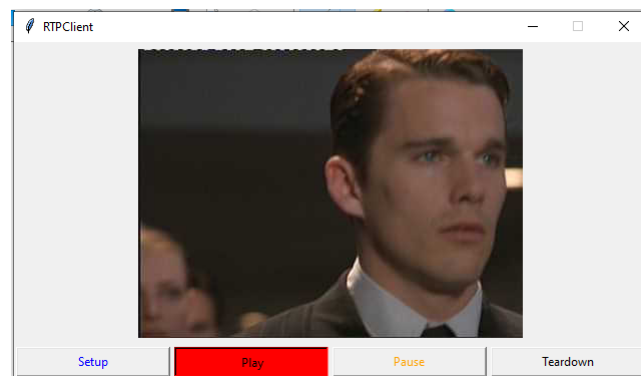


Figure 7: Giao diện khi người dùng bấm Play

- Khi chương trình đang chạy, nếu muốn tạm dừng thì bấm vào nút Pause

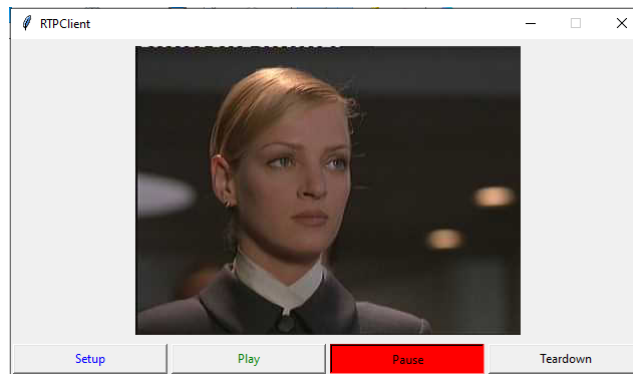


Figure 8: Giao diện khi người dùng bấm Pause

- Bấm vào Teardown thì chương trình sẽ dừng video và quay về trạng thái ban đầu chưa được SETUP.

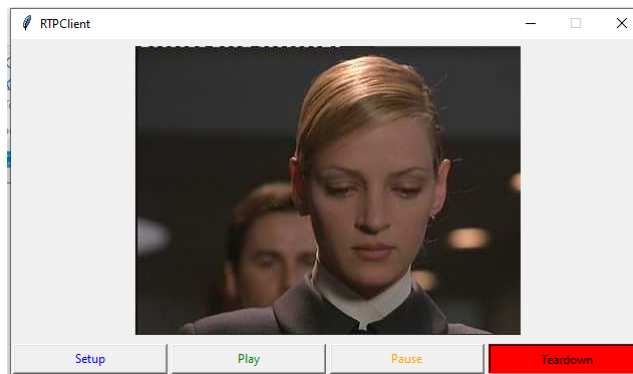


Figure 9: Giao diện khi người dùng bấm Teardown

- Để thoát khỏi chương trình, đóng cửa sổ giao diện người dùng, ta bấm vào dấu **X** ở góc trên phải màn hình. Bấm **OK** để thoát, bấm **Cancel** để hủy bỏ yêu cầu.

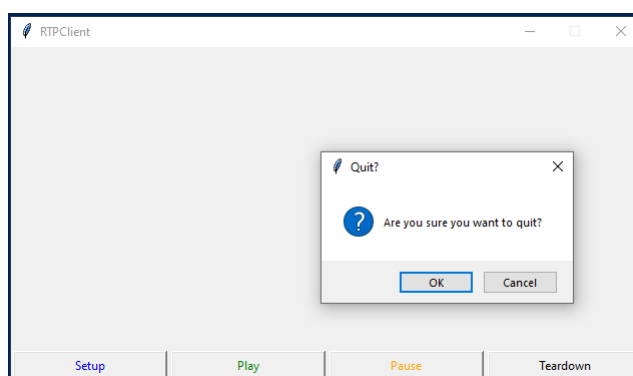


Figure 10: Giao diện khi bấm vào dấu X

6 Extend

6.1 Calculate the statistics about the session

a.) RTP PACKET LOSS RATE

Để xác định RTP packet loss rate ta có thể dùng các cách sau

- Cách 1: nếu dùng wire shark, ta có thể sử dụng RTP Stream Analysis

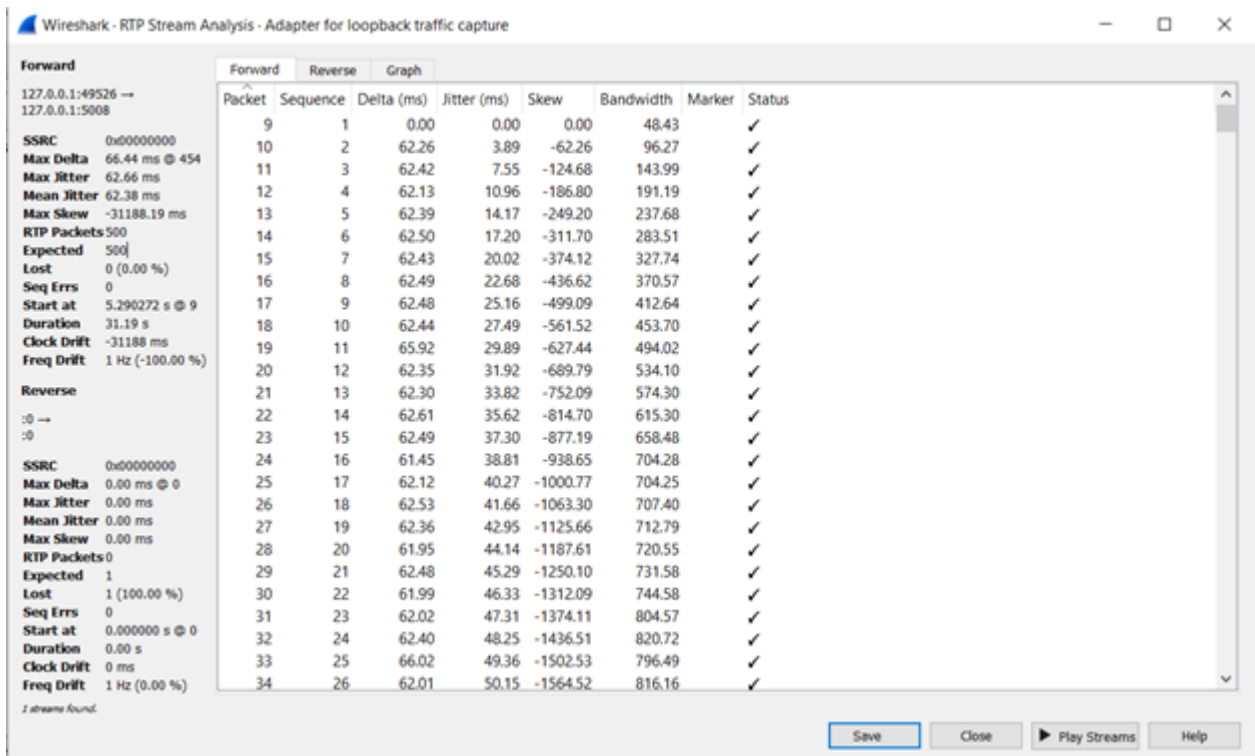


Figure 11: RTP Stream Analysis

Số RTP Packet được truyền là 500 (packets) và đã nhận đủ (không bị loss)

- Cách 2 : Xác định dựa trên sequence number ở bên phía Client, nếu như Current Sequence Number không tăng dần theo thứ tự thì chứng tỏ nó đã bị lost,

```

ServerWorker.py X
ServerWorker.py > ServerWorker > sendRtp
103
104     self.replyRtsp(self.OK_200, seq[0])
105
106     # Close the RTP socket
107     self.clientInfo['rtpSocket'].close()
108
109     def sendRtp(self):
110         """Send RTP packets over UDP."""
111         while True:
112             self.clientInfo['event'].wait(0.05)
113
114             # Stop sending if request is PAUSE or TEARDOWN
115             if self.clientInfo['event'].isSet():
116                 break
117
118             data = self.clientInfo['videoStream'].nextFrame()
119             if data:
120                 frameNumber = self.clientInfo['videoStream'].frameNbr()
121                 try:

```

Figure 12: thời gian đợi mỗi lần gửi tin

Theo như file ServerWorker.py, cứ mỗi 0.05 server sẽ gửi 1 gói tin, vậy thì tối đa có 20 gói được gửi, nhưng có những lúc mạng không ổn định thì có thể chỉ được 15-16 gói tin, vậy nên không thể xác định bằng cách đếm có đủ 20 gói tin trong 1s được.

Ta xử lý bằng cách đặt thêm biến counter khởi tạo bằng 0, biến này sẽ tăng mỗi khi listenRtp được gọi và bắt được data

```

def listenRtp(self):
    while True:
        try:
            data = self.rtpSocket.recv(20480)
            if data:
                rtpPacket = RtpPacket()
                rtpPacket.decode(data)

                currFrameNbr = rtpPacket.seqNum()
                self.counter += 1
                print("Current Seq Num: " + str(currFrameNbr))

                if currFrameNbr > self.frameNbr: # Discard the late packet
                    self.frameNbr = currFrameNbr
                    self.countPayload += len(rtpPacket.getPayload())
                    self.updateMovie(self.writeFrame(rtpPacket.getPayload()))

```

Figure 13: cộng current sequence number thêm 1 khi ListenRtp() được gọi

Khi hàm exitClient() được gọi, rate sẽ được tính bằng công thức: current frame number hiện tại (max frameNbr = 500) trừ đi số counter đã đếm được, đó chính là số packet bị loss, và chia cho frameNbr, chúng ta sẽ thu được tỷ lệ.

```
87  def exitClient(self):
88      """Teardown button handler."""
89      if self.state == self.READY and self.timeexe:
90          print("Video data rate = {0} / {1} = {2} bps". \
91                format(self.countPayload, self.timeexe, self.countPayload / self.timeexe))
92          self.sendRtspRequest(self.TEARDOWN)
93          #self.handler()
94          self.master.destroy() # Close the gui window
95      try:
96          os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT) # Delete the cache image from video
97      except:
98          pass
99      if self.frameNbr:
100         rate = float((self.frameNbr - self.counter)/self.frameNbr)
101         print('-'*60 + "\nRTP Packet Loss Rate : " + str(rate) + "\n" + '-'*60)
102         sys.exit(0)
103
```

Figure 14: Công thức tính Packet loss rate

b.) VIDEO DATA RATE

Về vấn đề video data rate, ta sẽ tính bằng tổng số Payload (chính là phần data trừ đi số byte của header) đếm được trong 1 đơn vị thời gian chia cho thời gian đã chạy video.

Ta xử lý bằng cách thêm thư viện time của python, 3 biến khởi tạo timestart, timeend, timeexe.

```
1  from tkinter import *
2  import tkinter.messagebox
3  from PIL import Image, ImageTk
4  import socket, threading, sys, traceback, os
5  import time
6  from RtpPacket import RtpPacket
7
8  CACHE_FILE_NAME = "cache-"
9  CACHE_FILE_EXT = ".jpg"
10
11 class Client:
12     INIT = 0
13     READY = 1
14     PLAYING = 2
15     state = INIT
16
17     SETUP = 0
18     PLAY = 1
19     PAUSE = 2
20     TEARDOWN = 3
21
22     countPayload = 0
23     counter = 0
24
25     timestart = 0
26     timeend = 0
27     timeexe = 0
28     # Initiation..
```

Figure 15: Thư viện time và các biến khởi tạo

Time start được đặt tính từ lúc người dùng bấm nút PLAY

```
111     def playMovie(self):
112         """Play button handler."""
113         if self.state == self.READY:
114             # Create a new thread to listen for RTP packets
115             self.timestart = time.time()
116             print("Playing Movie")
117             threading.Thread(target=self.listenRtp).start()
118             self.playEvent = threading.Event()
119             self.playEvent.clear()
120             self.sendRtspRequest(self.PLAY)
```

Figure 16: Timestart trong playMovie()

Timeend được set khi Pause được gọi, timeexe cũng được tính lúc này (bằng timeend – timestart), vì 1 video có thể bị pause nhiều lần, bởi Pause, hoặc nút thoát(X) nhưng lại cancel, nên chúng ta sẽ cộng dồn biến này lại

```
104     def pauseMovie(self):
105         """Pause button handler."""
106         if self.state == self.PLAYING:
107             self.timeend = time.time()
108             self.timeexe += self.timeend - self.timestart
109             self.sendRtspRequest(self.PAUSE)
110
```

Figure 17: Timeexe and Timeend trong pauseMovie()

Tổng số Payload thì được cộng dồn mỗi khi listenRtp được gọi và bắt được data

```
122     def listenRtp(self):
123         while True:
124             try:
125                 data = self.rtpSocket.recv(20480)
126                 if data:
127                     rtpPacket = RtpPacket()
128                     rtpPacket.decode(data)
129
130                     currFrameNbr = rtpPacket.seqNum()
131                     self.counter += 1
132                     print("Current Seq Num: " + str(currFrameNbr))
133
134                     if currFrameNbr > self.frameNbr: # Discard the late packet
135                         self.frameNbr = currFrameNbr
136                         self.countPayload += len(rtpPacket.getPayload())
137                         self.updateMovie(self.writeFrame(rtpPacket.getPayload()))
```

Figure 18: Timeexe and Timeend in pauseMovie()

Khi exitClient() được gọi, Video data rate sẽ được tính bằng công thức tổng số Payload đếm được chia cho thời gian từ lúc chạy tới lúc kết thúc (không tính thời gian PAUSE) việc gửi các gói tin.

```
87     def exitClient(self):
88         """Teardown button handler."""
89         if self.state == self.READY and self.timeexe:
90             print("Video data rate = {0} / {1} = {2} bps". \
91                   format(self.countPayload, self.timeexe, self.countPayload / self.timeexe))
92             self.sendRtspRequest(self.TEARDOWN)
93             #self.handler()
94             self.master.destroy() # Close the gui window
95             try:
96                 os.remove(CACHE_FILE_NAME + str(self.sessionId) + CACHE_FILE_EXT) # Delete the cache image from video
97             except:
98                 pass
99             if self.frameNbr:
100                 rate = float((self.frameNbr - self.counter)/self.frameNbr)
101                 print('-'*60 + "\nRTP Packet Loss Rate : " + str(rate) + "\n" + '-'*60)
102             sys.exit(0)
```

Figure 19: countPayload in listenRtp()

Khi exitClient() được gọi, Video data rate sẽ được tính bằng công thức tổng số Payload đếm được chia cho thời gian từ lúc chạy tới lúc kết thúc (không tính thời gian PAUSE) việc gửi các gói tin.

```
138
139         except:
140             # Stop listening upon requesting PAUSE or TEARDOWN
141             if self.state == self.PLAYING:
142                 self.pauseMovie()
143                 print('-'*60 + "\nLast packet is received!!!" + "\n" + '-'*60)
144                 print("Didn't receive data!")
145                 if self.playEvent.isSet():
146                     break
147
148             # Upon receiving ACK for TEARDOWN request,
149             # close the RTP socket
150             if self.teardownAcked == 1:
151                 self.rtpSocket.shutdown(socket.SHUT_RDWR)
152                 self.rtpSocket.close()
153                 break
```

Figure 20: Video data rate formula

Đây là kết quả khi chạy hết video. Với bps là bytes per second.


```
TERMINAL  OUTPUT  DEBUG CONSOLE
Current Seq Num: 489
Current Seq Num: 490
Current Seq Num: 491
Current Seq Num: 492
Current Seq Num: 493
Current Seq Num: 494
Current Seq Num: 495
Current Seq Num: 496
Current Seq Num: 497
Current Seq Num: 498
Current Seq Num: 499
Current Seq Num: 500
-----
PAUSE request sent to Server...
-----
Last packet is received!!!
-----
Parsing Received Rtsp data...
Didn't receive data!
Video data rate = 4267393 / 32.431803464889526 = 131580.5025958502 bps
-----
TEARDOWN request sent to Server...
-----
RTP Packet Loss Rate :0.0
-----
Parsing Received Rtsp data...
```

Figure 21: Output with RTP Packet loss rate and Video data rate

6.2 Implement PLAY, PAUSE and STOP buttons

SETUP là bắt buộc trong tương tác RTSP, ở trong bài tập lớn này, chúng ta có 2 cách đơn giản để hiện thực nó:

- Cách 1: Tự động SETUP ngay sau khi gõ lệnh ở terminal thứ 2.
- Cách 2: Chương trình sẽ SETUP ở lần bấm PLAY đầu tiên.

Ở đây, nhóm chúng em xin chọn cách 1 để hiện thực nó.

Trong Windows Media Player, khi ta bấm vào STOP thì video sẽ dừng và quay trở về trạng thái ban đầu và đã được SETUP sẵn. Khi đó nếu bấm PLAY thì video sẽ chạy lại từ đầu. Ta chỉ có thể thoát khi bấm vào dấu X ở góc bên phải phía trên của ứng dụng.

Mã nguồn là ở folder **Extend 2** hoặc link github của nhóm <https://github.com/KhangSK/Computer-Network/tree/master/Extend%202>

Dưới đây là các bước thao tác cơ bản

- Sau khi gõ 2 lệnh lần lượt trên 2 terminal. Chương trình sẽ tự động SETUP sẵn.

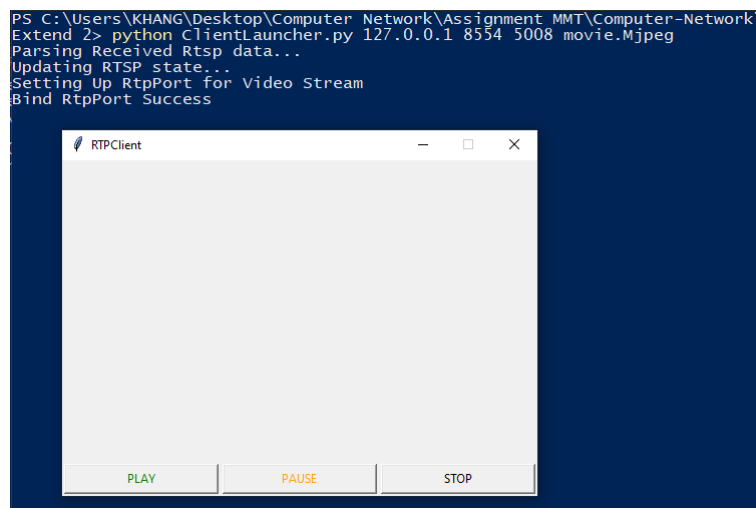


Figure 22: Chương trình tự động SETUP

- Khi bấm PLAY, video sẽ được chạy.

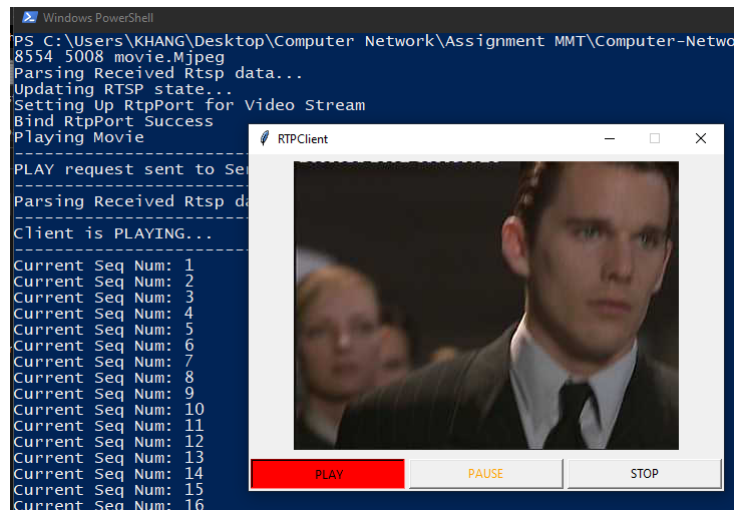


Figure 23: Chạy video bằng bấm vào nút PLAY

- Khi bấm PAUSE, video sẽ tạm dừng

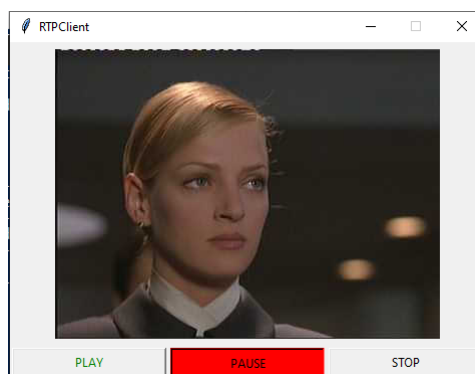


Figure 24: Video đang tạm dừng

- Khi người dùng bấm vào STOP, thì video sẽ dừng lại và quay trở về trạng thái ban đầu. Được SETUP sẵn tương tự như Windows Media Player.

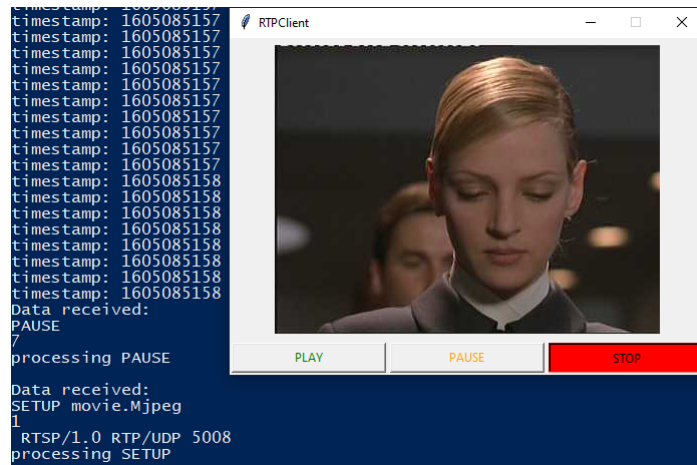


Figure 25: Khi bấm STOP sẽ quay trở lại trạng thái ban đầu

- Và cuối cùng, để thoát khỏi chương trình thì người dùng bấm vào dấu X ở góc phải màn hình. Chương trình sẽ thoát ngay lập tức mà không hỏi lại (tương tự Windows Media Player)



Figure 26: Thoát chương trình

Với cách hiện thực như trên thì ta nhận thấy, khi bấm vào TEARDOWN ở phần hiện thực trong Bài tập lớn thì video sẽ dừng lại và quay về trạng thái ban đầu chưa được SETUP. Còn đối với STOP hiện thực ở trong phần extend này thì video sẽ dừng lại và quay về trạng thái ban đầu và được SETUP sẵn.

6.3 Implement DESCRIBE method

Ta tạo thêm 1 nút DESCRIBE trong giao diện người dùng và đồng thời viết thêm hàm để truyền yêu cầu DESCRIBE đến server. Yêu cầu DESCRIBE chỉ được gửi đi khi người dùng đã SETUP thành công hoặc chương trình đang trong trạng thái READY hoặc PLAYING.

```
290
291     # Describe request
292     elif requestCode == self.DESCRIBE and not self.state == self.INIT:
293         # Update RTSP sequence number.
294         self.rtpSeq = self.rtpSeq + 1
295
296         # Write the RTSP request to be sent.
297         request = "DESCRIBE " + "\n" + str(self.rtpSeq)
298         self.rtpSocket.send(bytes(request, 'utf-8'))
299         print('-'*60 + "\nDESCRIBE request sent to Server...\n" + '-'*60)
300
301         # Keep track of the sent request.
302         self.requestSent = self.DESCRIBE
303
```

Figure 27: Viết hàm xử lý yêu cầu Describe từ Client

Khi Server đã nhận được yêu cầu DESCRIBE qua giao thức RTSP, ta sẽ viết đoạn xử lý yêu cầu và gọi hàm replyDescribe để trả về thông tin của media stream.

```
182     def replyDescribe(self, code, seq):
183         """Send RTSP Describe reply to the client."""
184         if code == self.OK_200:
185             #print("200 OK")
186             reply = 'RTSP/1.0 200 OK\r\nCSeq: ' + seq + '\r\nSession: ' + str(self.clientInfo['session'])
187
188             descriptionBody = "\n\nv=0"
189             descriptionBody += "\nm=video " + self.clientInfo['rtpPort'] + " RTP/AVP " + str(MJPEG_TYPE)
190             descriptionBody += "\na=control:streamid=" + str(self.clientInfo['session'])
191             descriptionBody += "\na=mimetype:string;\nvideo/MJPEG\r\n"
192
193             reply += "\n\nContent-Base: " + self.clientInfo['VideoFileName']
194             reply += "\nContent-Type: " + "application/sdp"
195             reply += "\nContent-Length: " + str(len(descriptionBody))
196             reply += descriptionBody
197
198
199             connSocket = self.clientInfo['rtpSocket'][0]
200             connSocket.send(reply.encode())
201
```

Figure 28: Tạo hàm replyDescribe để tạo thông điệp trả về Client

Cổng mặc định được sử dụng cho giao thức RTSP là 554 và cổng này được sử dụng cho cả giao thức của tầng giao vận UDP và TCP.

Thông điệp trả về theo SDP format (Session Description Protocol) đây là format dùng để mô tả các thông tin phiên giao tiếp của các chương trình multimedia. Ngoài ra trong thông điệp trả về từ máy server còn liệt kê các đường link thích hợp hơn tới file video cần chơi khi mà trong file video đó có trộn lẫn giữa phụ đề và âm thanh. Và điều quan trọng nhất ở trong bản tin miêu tả phiên giao dịch này là streamid của luồng video và streamid của luồng âm thanh khi mà đoạn video đó có lồng âm thanh vào trong các frame.

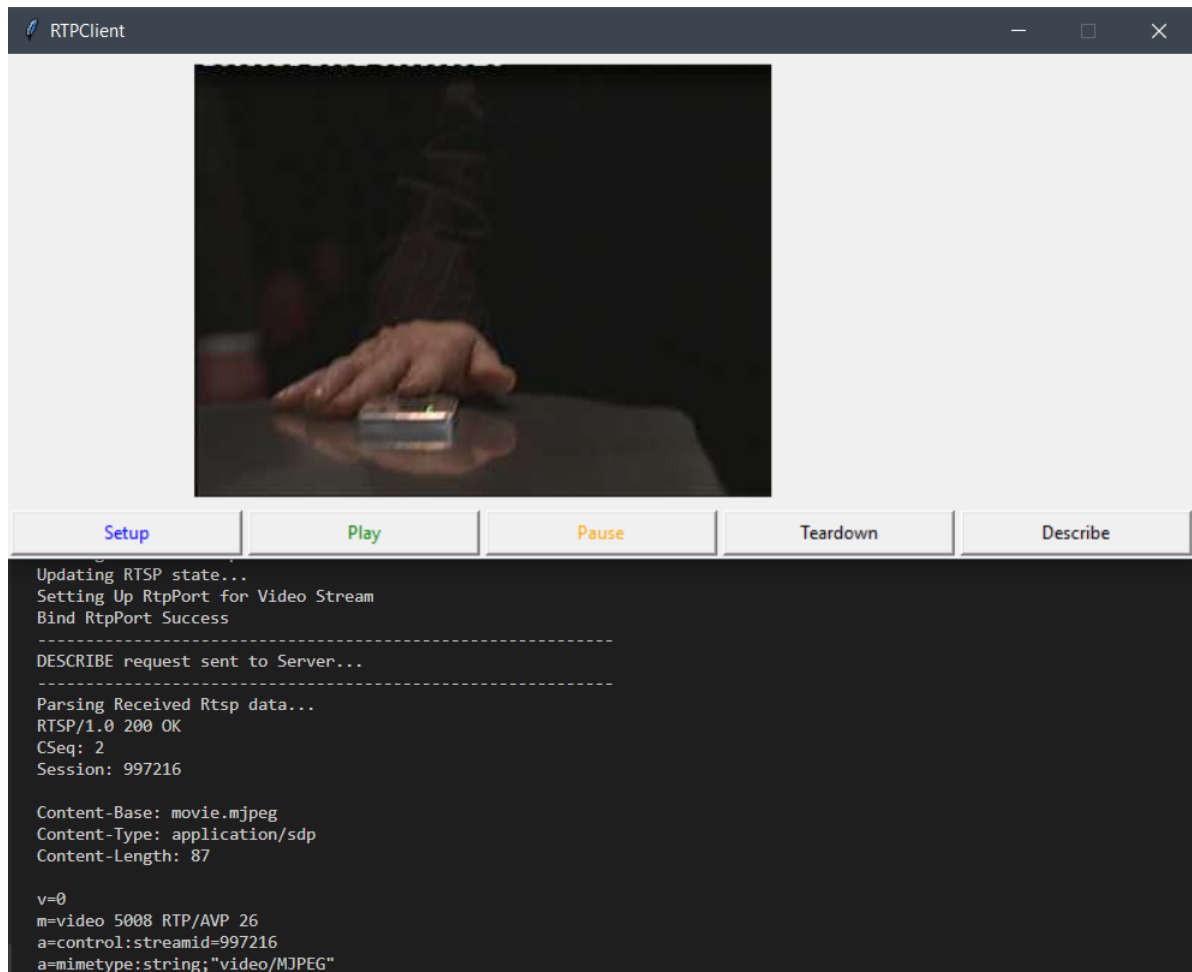


Figure 29: Thông điệp trả về khi người dùng ấn Describe

Nội dung thông điệp trả về từ yêu cầu Describe của client như hình trên. Trong đó, phần nội dung thông điệp chứa những thông tin sau:

- $v=0$: phiên bản của giao thức, hiện chỉ là 0
- $m=video\ 5008\ RTP/AVP\ 26$: kiểu của chương trình media và địa chỉ transport.
Theo đặc tả của SDP, 'm' line format theo dạng sau:
 $m=<media>\ <port>\ <transport>\ <fmt>$
Trong đó, kiểu $<media>$ là video, phần $<port>$ là RTPport, $<transport>$ là RTP/AVP, và $<fmt>$ là MJPEG_TYPE = 26
- $a=control:streamid=997216$: streamid chính là session id của video stream.
- $a=mimetype:string;"video/MJPEG"$: Kiểu phương tiện (MIME type hay media type) là định danh hai phần cho định dạng file và nội dung định dạng được truyền trên Internet.
"video/MJPEG" cho biết kiểu nội dung đang streaming trên session là video và file được mã hóa bằng MJPEG.

7 Reference

References

- [1] Real Time Streaming Protocol (RTSP) , truy cập từ: <https://tools.ietf.org/html/rfc2326>
- [2] Software Requirements Analysis with Example , truy cập từ: <https://www.guru99.com/learn-software-requirements-analysis-with-case-study.html>
- [3] RTP: A Transport Protocol for Real-Time Applications, truy cập từ: <https://tools.ietf.org/html/rfc1889>
- [4] Session Description Protocol (SDP) Format for Real Time Streaming Protocol (RTSP) Streams, truy cập từ: <https://tools.ietf.org/html/draft-marjou-mmusic-sdp-rtsp-00>
- [5] Session Description Protocol, truy cập vào 11/11/2020 từ: https://en.wikipedia.org/wiki/Session_Description_Protocol